Julia Language 1.1 Ephemeris Reader and Gravitational Modeling Program for Solar System Bodies

Kaela Martin[1], Tristan Minkoff [2], Parker Landon[3], Brennan Gray[4]

College of Engineering, Embry-Riddle Aeronautical University, Prescott, AZ 86301, USA

and

Damon Landau[5]

Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA 91109-8099, USA

This paper analyzes the advancements to the Julia Language 1.1 Ephemeris and Physical constants Reader including the addition of gravitational modeling. Originally written in MATLAB, this Julia Language program is intended to be used in for trajectory design. Written in an open-source coding language, this ephemeris reader can output the state of planetary bodies including asteroids as well as other constants such as gravitational parameters. Two primary methods were chosen to calculate the gravitational potentials which include polyhedral modeling and spherical harmonics.

I. Introduction

Designing space trajectory missions requires an abundance of data over large lengths of time. An ephemeris reader can generate this data by interpreting files containing the positions and velocities of objects in space. Ephemeris readers have not only helped with trajectories [1, 2], but spacecraft telemetry [3] and satellite orbits [4] as well.

The Julia Language was chosen due to its ability to handle large computations while still being a relatively simple software. First released in 2012, Julia is a dynamic language possessing similar attributes to both MATLAB and C. Julia is used primarily for computationally strenuous endeavors [5]. The robustness of the language is complemented by the user-friendly syntax, and the first stable version, Julia 1.1, was released in August 2019[*]. Due to Julia's previously unstable pre-1.0 state, the project's priority was to transfer the Ephemeris reader to Julia 1.1 and produce an operating function call, boddat.

Julia provides the accuracy and speed necessary for an ephemeris tool. Although one documented ephemeris reader exists in Julia, JPLEphemeris.jl, this reader purely calculates the positions and velocities of the major bodies within our solar system[†]. The ephemeris reader discussed in this paper can calculate both major and small body objects (i.e., asteroids). The initial program was developed in MATLAB and was then moved to Julia and improved upon [6, 7]. This project is a continuation from last year's.

---

[1] Assistant Professor, Aerospace Engineering, Embry-Riddle Aeronautical University, Prescott, AZ 86303,AIAA and AAS Member
[2] Student, Aerospace Engineering, Embry-Riddle Aeronautical University, Prescott, AZ 86303, AIAA Member
[3] Student, Computer Engineering and Space Physics, Embry-Riddle Aeronautical University, Prescott, AZ 86303,AIAA Member
[4] Student, Aerospace Engineering, Embry-Riddle Aeronautical University, Prescott, AZ 86303,AIAA Member
[5] Mission Formulation Engineer, Project Systems Engineering and Mission Formulation Group, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA 91109, USA.

[*] Data available online at https://julialang.org/ [Accessed 15 Nov. 2019]
[†] Data available online at https://github.com/JuliaAstro/JPLEphemeris.jl [Accessed 15 Nov. 2019]

NASA's Jet Propulsion Laboratory has developed several ephemerides in their Development Ephemeris (DE) series with DE3 and DE19 as two of the earliest ephemerides [8, 9]. They construct data by integrating differential equations of motion but only supported solar system major bodies [10]. Advancements through 1998 provided more objects including the major planets, the Sun and three of the largest known asteroids. As space travel progressed, so did the ephemeris. Recent Ephemerides include NASA's Horizons[‡], NAIF[§], and Small Body Databases[**]. Utilizing these databases, the ephemeris reader is able to output the locations of major and small bodes.

Gravitational modeling is also an important aspect of building trajectories. With recent missions like OSIRIS-Rex [12], gravitational potentials of irregularly shaped objects are becoming a necessity. Several methods can be used to calculate these potentials. Current modeling approaches for gravitational potentials include polyhedral, spherical, and ellipsoidal harmonics [12, 13]. This paper focuses on two main solutions, polyhedral modeling and spherical harmonics, and one fast approximation referencing the Brillouin Sphere. Polyhedral modeling accesses the Database of Asteroid Models from Inversion Techniques (DAMIT)[††]. DAMIT manages thousands of asteroid properties and models. The ephemeris reader, with its subfunctions, use these models to reproduce the gravitational fields of the asteroids.

II. Implementation

This project focused on improving the capabilities of the ephemeris reader in the Julia Language as well as converting the ephemeris reader to Julia 1.1, now a stable version of Julia. The original ephemeris reader, called boddat, was written in MATLAB by the last author. Since the start of this project, boddat was converted over to the Julia Language from MATLAB to increase the computational efficiency. Once boddat was converted over to Julia and transitioned to the most recent version of Julia, new features were implemented including spherical harmonics, or gravmod, polyhedral harmonics, or polymod, and asteroid shape modeling, or astermod.

By implementing the gravitational modeling functions to the ephemeris reader, the user now has the capability to perform high level mission planning for asteroid rendezvous missions. Similar missions such as OSIRIS-REx and Hayabusa2 could have used this ephemeris reader as a toolbox during the early mission design phases. This toolbox allows the user to obtain data to assist in modeling potential orbits around various asteroids as well as rendezvous trajectories to the surface of various asteroids.

A. Ephemeris Reader

The version presented in this paper is the third iteration of the ephemeris reader. The goals of this version were decreased execution time and more robust handling of unexpected input. Additionally, a Julia package was created to ease the installation and usage of the program.

Several techniques were employed to decrease the execution time of the code, with a focus on functionality that would be called in batches. The simplest of these techniques was a reduction in memory allocations made by the program. As memory in modern computer systems is generally significantly slower than the processor, any retrieval from memory has significant performance overhead. As the ephemeris reader uses large arrays and dictionaries to store data during execution, there is a significant performance overhead associated with memory allocation and access. While some allocations are unavoidable, many were unnecessary, particularly in the case of array copying. However, Julia provides an easy method of avoiding copying. Consider the following code

```
B = A[1, :]
```

This code creates an array B, which contains the first row of array A. In Julia, array B would be a copy of the requested data of array A, stored elsewhere in memory. Copying has the advantage of making the data stored in

[‡] Data available online at https://ssd.jpl.nasa.gov/horizons.cgi [retrieved 15 Nov. 2019]
[§] Data available online at https://naif.jpl.nasa.gov/naif/ [retrieved 15 Nov. 2019]
[**] Data available online at https://ssd.jpl.nasa.gov/sbdb.cgi [retrieved 15 Nov. 2019]
[††] Data available online at https://astro.troja.mff.cuni.cz/projects/asteroids3D/web.php [retrieved 15 Nov. 2019]

arrays A and B independent of each other; however, if array B is never modified this advantage is not useful. Julia provides the view function for such situations

```
B = view(A, 1, :)
```

This code instead makes B a reference to the data in A, sharing the same part of the memory. While the same effect could be accomplished through using A[1,:] in place of B, use of the view function also has the benefit of making code more readable. In this version of the ephemeris reader, the view function is used in place of array copying where possible, most notably in the spline evaluation code which forms the core of the ephemeris interpolation functions.

Another major decrease in execution time came from the restructuring of the code to avoid unnecessary overhead. In the previous version of the ephemeris reader, all ephemeris data was accessed through the boddat function. Since the boddat function could redirect to any number of internal functions, some time was spent on parsing the user's input and converting it into a function call. The boddat function also saved the dictionary used to store data between executions every time it was executed. Since modern mass storage devices are extremely slow relative to the processor and memory speeds, much of the execution time of boddat was spent saving data.

To alleviate the overhead associated with the boddat-centered use structure, much of the code was re-architected to allow users to directly call the required subfunctions. Migration away from boddat resulted in an order-of-magnitude gain in performance. For a comparison, consider the form of a previous boddat function call

```
boddat([“R”], [399], dict)
```

This function call would retrieve the position of Earth relative to the Solar-System Barycenter at the current time. In the new, subfunction-driven structure, a call to a subfunction would be made instead:

```
ephem(399, Dates.datetime2julian(now()) - 2451545., EphemType.position, dict)
```

Like the previous boddat call, the body, ephemeris type, and a dictionary must be provided. However, unlike boddat, these subfunctions do not allow for calls without a specified input time, which is provided as the second argument. The time must be a Julian date in the J2000 epoch.

These subfunctions do not handle many of the tasks boddat once handled to improve performance. Since users must manually perform many of the tasks that boddat once handled, documentation was created to guide the users through proper use of the code. The boddat function was left available for applications that are not performance centric.

As users are now encouraged to directly call subfunctions, there is far more variability of input to these subfunctions that must now be accounted for. Several steps were taken to account for this variability. Many arguments were type annotated to ensure that they could be properly handled. As Julia is a dynamically typed language, it does not require type annotations for function arguments to execute properly and will automatically generate versions of functions to handle different input argument types. However, the subfunctions of the code are not robust enough to handle all types of input, and many input types do not make sense in the contexts of the functions. For example, consider the following function declaration:

```
position(body, time)
```

Ideally, this function returns the position of a body at a given time. However, just from this declaration it is impossible to know what the types of the input must be. The body argument could be an integer, representing a SPK ID, or a string. Type annotating functions alleviates this problem:

```
position(body::Integer, time::Float)
```

In this example, it is clear that the body argument must be an integer. Type annotating the arguments of functions restricts what inputs the user can provide to types that can be properly processed. Julia will also give users suggestions on input types that are available when provided with invalid input, giving users some guidance on how to properly call the function they are trying to use.

Additionally, code to check the inputs was inserted into many functions to check for inputs that are type correct but invalid. Accompanying this change are more descriptive error messages and warnings to make the user aware of how their input is invalid. Many new error messages and warnings were also added in the code itself in case of unexpected behavior. For example, consider the following function call:

```
gettype(399)
```

This function call would attempt to retrieve the spectral type for Earth. Since Earth is a major body, it does not have a spectral type. Previously, this function call would fail without any explanation. Instead, it now gives a descriptive warning message of the problem:

```
Warning: Cannot retrieve spectral type of major body 399
```

Descriptive error and warning messages like this message give the user an explanation of why a function call failed instead of leaving them to debug the problem without any indication of why the failure occurred.

Additionally, these new errors and warnings defend against formatting changes to the data sources on which the code relies. Any changes to the data sources would previously result in an unexplained failure of the data retrieval code, which was very difficult to debug. Instead, the current version gives descriptive error messages indicating that formatting changes have occurred.

Lastly, the code was placed in a Julia package which can easily be installed through the Julia terminal. Unlike the previous version of the ephemeris reader, the user no longer has to declare the dependencies of the code at the start of their own code, instead issuing a single using statement for the new package which incorporates all dependencies automatically.

Currently boddat is able to provide the user with state data for small bodies. However, mission planning may require more information than the location of the asteroid. Therefore, a separate subfunction was created to assist in gathering various properties pertaining to asteroids, this is discussed in the next section.

B. Asteroid Property Identification

The asteroid property identification function, astermod, is able to provide the user with specific properties of user requested asteroids by locating and parsing the information from the DAMIT database. For example, the input is similar to

```
astermod([“asteroid 1” “asteroid 2”],[“property 1” “property 2” “property 3”])
```

where “asteroid 1” and “asteroid 2” are the specific asteroids that the user is calling, and “property 1” through “property 3” are the specific properties that the user is requesting. The example input for astermod illustrates the user calling two asteroids and three properties. However, the user can call as many asteroids and properties that they may require. Both the asteroid names and property names must be input as strings, if the user does not input the values as strings the function will output text similar to

```
“Not a valid input”
```

Once the user properly inputs the requested asteroids and the properties, astermod will output the values as a matrix. An example of astermod calling asteroids Thalia and Europa, along with properties beta, equivalent diameter, and period can be shown

```
astermod([“Thalia” “Europa”],[“beta” “equivalent diameter” “period”])
```

The input would yield the following results

```
[“asteroid id” “model id” … “equivalent diameter” “period”]
[“===“, “===“, “===“, “===“, “===“]
[“115”, “122”, “-45”, “107”, “12.31241”]
[“115”, “123”, “-69”, “107”, “12.31241”]
[“115”, “1857”, “-46”, “115”, “12.31241”]
[“115”, “1858”, “-74”, “114”, “12.31241”]
[“===“, “===“, “===“, “===“, “===“]
[“125”, “135”, “35”, “293”, “5.629958”]
[“125”, “463”, “39”, “319”, “5.629959”]
[“===“, “===“, “===“, “===“, “===“]
```

The first row of the output acts as a key that informs the user on what each column of the output contains. The row of “===” is an indication of a change of output. For example, the second row separates the key from the output of the first asteroid, while the seventh row separates the output from the first asteroid from the output of the second asteroid. The output also shows the model ID associated with each asteroid. The model ID is the asteroid’s model identification number. Therefore, astermod will output a number of rows for a specific asteroid that is equivalent to the number of model IDs associated with that specific asteroid. Shown in the output, Thalia has four separate model IDs; therefore, Thalia has four sets of properties corresponding with each model ID.

When planning a mission to an asteroid, the user must identify the trajectory of the asteroid, any necessary properties such as equivalent diameter, and the gravitational field of the asteroid. The user is able to obtain both the trajectory and the properties of asteroids from boddat and astermod respectively. To account for the third piece of information that the user will require for asteroid mission planning, the gravitational field of the asteroids, three different methods to calculate the gravitational field are discussed in this paper. The first method utilizes the DAMIT database, in order to download asteroid shape files to construct a polyhedron model, which is discussed further in the next section.

C. Polyhedron Gravitational Modeling

The first approach to modeling the gravitational field of a body discussed in this paper is the constant-density polyhedron method. The polyhedron approach allows the user to model various geometries including concavities in its surface (craters), overhangs, interior voids (caves), and holes all the way through the body. The accuracy of the polyhedron approach is only limited by the size of the mesh of the body. Therefore, the gravitational field is exact for the given shape and density of the body. Two assumptions associated with the polyhedron approach are that (1) the body has a constant density and (2) the body is a polyhedron.

To model the gravitational potential of a body, the function downloads a text file from the DAMIT database containing the locations of each of the polyhedron’s vertices and their connective topology. For example, the asteroid Kleopatra was modeled as a polyhedron using the information from the DAMIT database, shown in Fig. 1.

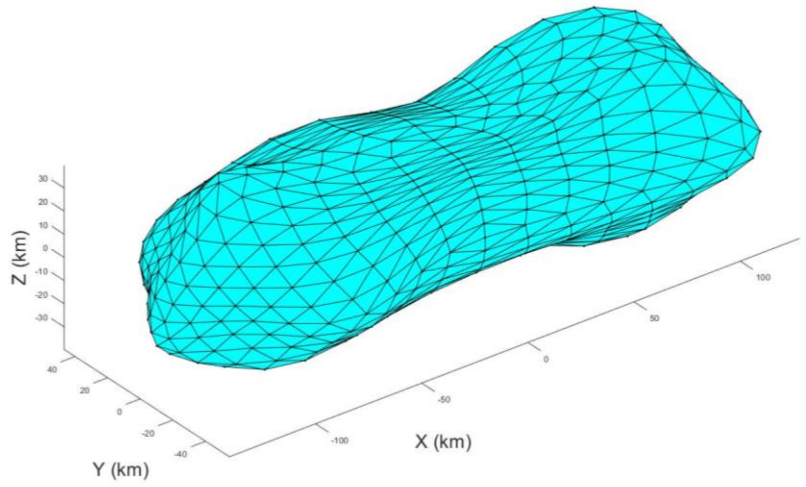

Fig. 1: Polyhedron model of Kleopatra

By using the geometric information from the DAMIT database the function is able to calculate the gravitational potential, attraction, and the Laplacian of every asteroid body available in the database. The polyhedron method minimizes the error by using the information from the DAMIT database to create a mesh that converges on the surface of the body, which is shown in Fig. 1. The following formulation to calculate the gravitational potential, attraction, and the Laplacian was used from Werner and Scheeres [12]. The equation used to calculate the gravitational potential is represented by

$$U = \frac{1}{2}G\sigma \sum_{e \in edges} \overline{r}_e E_e \overline{r}_e L_e - \frac{1}{2}G\sigma \sum_{f \in faces} \overline{r}_f F_f \overline{r}_f \omega_f \tag{1}$$

where U is the gravitational potential, G is the gravitational constant, σ is the bulk density, e as a subscript represents the specific edge in each iteration, $\overline{r_e}$ is a vector from the field point to any point on edge e, E_e is the dyad associated with edge e and face f, L_e is the potential of a 1D straight wire for edge e of face f, f as a subscript represents the specific face in each iteration, $\overline{r_f}$ is a vector from the field point to any point on face f, F_f is the dyad associated with face f, and ω_f is the signed area of face f projected onto the unit sphere centered at the field point. The equation used to calculate the dyad associated with edge e and face f is represented by

$$E_{12} = \hat{n}_A \hat{n}_{12}^A + \hat{n}_B \hat{n}_{21}^B \tag{2}$$

where A as a subscript represents the specific plane of interest in each iteration, B as a subscript represents the plane adjacent to plane A, $\hat{n}_A$ is the normal vector facing outwards from plane A, $\hat{n}_{12}^A$ is the normal edge vector which is perpendicular to the edge and points outwards, $\hat{n}_B$ is the normal vector facing outwards from plane B, and $\hat{n}_{12}^B$ is the normal edge vector which is perpendicular to the edge and points outwards from plane B. These vectors can be seen in Fig. 2.

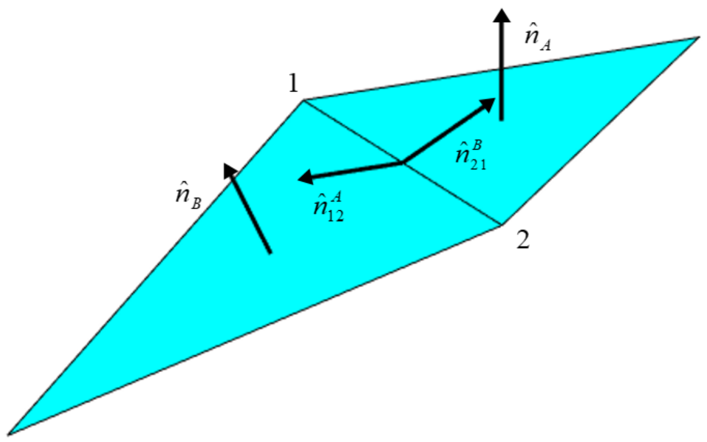

Fig. 2: Two face planes and their face normal and edge normal vectors Based on Werner and Scheeres [13]

Equation 2 contains the subscripts 12 and 21. These subscripts indicate the edge that is connecting the two surfaces and the orientation the edge is in with relation to the surface. The edge connecting the two surfaces in Fig. 2 is indicated by vertices 1 and 2. Therefore, for this specific edge, E_e is equal to E_{12}. The edge dyad, E_e, is the sum of two outer products and creates a symmetric 3 x 3 matrix. The face dyad, F_f, is calculated by taking the outer product of each face vector with itself. The face dyad is represented by

$$F_f = \hat{n}_f \hat{n}_f \tag{3}$$

The face dyad, like the edge dyad, is also a symmetric 3 x 3 matrix. The equation used to calculate the potential of a 1D straight wire is represented by

$$L_e = \ln\left(\frac{a+b+ed}{a+b-ed}\right) \tag{4}$$

where a represents the distance between the field point and one end of the edge e, b represents the distance between the field point and the other end of edge e, and ed is the edge length. The signed area of face f projected onto the unit sphere centered at the field point is represented by

$$\omega_f = 2\arctan\left(\frac{\bar{r}_1 \cdot \bar{r}_2 \times \bar{r}_3}{r_1 r_2 r_3 + r_1(\bar{r}_2 \cdot \bar{r}_3) + r_2(\bar{r}_3 \cdot \bar{r}_1) + r_3(\bar{r}_1 \cdot \bar{r}_2)}\right) \tag{5}$$

The original ω_f equation has been simplified due to the assumption that the polyhedron is a triangle. The function is able to utilize this assumption because the polyhedron's vertices and their connective topology from the DAMIT database form triangular surfaces. The subscripts 1, 2, and 3 correspond with vertices of each of the triangular surfaces, and $\bar{r}$ is the vector from the field point to the corresponding vertex. The 3D polyhedron attraction vector is represented by

$$\nabla U = -G\sigma \sum_{e \in edges} E_e \bar{r}_e L_e + G\sigma \sum_{f \in faces} F_f \bar{r} \omega_f \tag{6}$$

The polyhedron attraction vector points in the direction of the body while its magnitude is the gravitational acceleration experienced at the location of the field point. In order to determine whether the field point is inside or outside the asteroid, the function calculates the Laplacian which is represented by

$$\nabla^2 U = -G\sigma \sum_{f \in faces} \omega_f \tag{7}$$

When the field point is inside the polyhedron, the Laplacian is equal to -4π, or Poisson's equation. However, when the field point is outside the polyhedron, the Laplacian is equal to 0, or Laplace's equation. The Laplacian calculation comes at little to no cost because ω_f is already calculated in both the gravitational potential and the attraction equations.

The primary output for this function is the 3D polyhedron attraction vector. To obtain the attraction vector the user can either input the field point with respect to the asteroid or the user can input both the field point and the asteroid with respect to another body. The user can specify an asteroid by using the asteroid name, the SPK-ID, or the DAMIT database asteroid ID. The implementation of the function to the main ephemeris is currently ongoing and is projected to be completed within the upcoming months.

Although gravitational fields of small bodies can be calculated using polymod, many asteroids do not have the necessary shape files to construct the polyhedron shape. Therefore, a function, gravmod, was created to calculate the potentials of these bodies using spherical harmonic expansions, which is discussed in the next section.

D. Spherical Harmonics

Spherical harmonics are frequency-space representations of functions defined over a sphere. Spherical harmonics provide insight to electron configurations, magnetic fields, and solutions for the Schrödinger equation[‡‡]. They are also useful to represent gravitational fields of planetary bodies. The previous version of gravmod calculated to the fourth-degree polynomial. The objective of this project was to surpass this limit and pursue higher orders of harmonics. Due to the computational complexity, most harmonic solutions can be calculated up to the 150th degree before over and under-flow [14].

The spherical harmonic function, gravmod, produces the potential by way of harmonic expansions using specific user inputs.

```
potentialB = gravmod([“body”], [Pos], [t])
```

where ‘body’ is the reference string to the requested body, ‘Pos’ is the spacecraft's position in kilometers with respect to the Sun in Cartesian coordinates, and ‘t’ is the defined time in days past non in TDB on J2000.

For example, the inputs to find potentials around Earth would be similar to

```
gravmod([“Earth”],[-1.5*10^7,1.5*10^8,-7*10^3])
```

providing the output

$$6.91e8 \ \frac{\text{km}^2}{\text{s}^2}$$

which is the potential on the spacecraft at the user defined location.

Using converging spherical harmonic expansions, and an infinite series parameterized in spherical coordinates, (r, ϕ, θ), the gravitational potential for both major and small bodies can be represented by

[‡‡] Data available online at https://cs.dartmouth.edu/~wjarosz/publications/dissertation/appendixB.pdf [retrieved 15 Nov. 2019]

$$U = \frac{GM}{R_b}\sum_{n=0}^{\infty}\left(\frac{R_b}{r_c}\right)^{n+1}\sum_{m=0}^{n}\left[a_{nm}\left(\cos\left(\theta\right)\right)\cos\left(m\phi\right)+b_{nm}P_n^m\left(\cos\left(\theta\right)\right)\sin\left(m\phi\right)\right] \quad (8)$$

where *G* is the gravitational constant, *M* is the reference body mass, R_b is the reference body radius, r_c is the spacecraft's location from the body, *n* is the degree, *m* is the order of expansion, P_n^m is the normalized Legendre polynomials, and a_{nm} and b_{nm} are the harmonic coefficients.

The radius, prime meridian, and gravitational parameter are procured from the boddat function itself to determine the potential. Because the latitude of the spacecraft is measured from the prime meridian of the body, the prime meridian direction cosine matrix (DCM) will be used. This matrix is provided from boddat as the transformation from the body-fixed frame to EMO2000, which is the frame of the spacecraft position input. Additionally, the latitude, longitude, and the spacecraft's position, r_c, can be determined using the DCM. The spacecraft's position is with respect to the reference body, solved by both positions in respect of the Sun. The function call requires the user to input the spacecraft with respect to the Sun as well as the time at which both the body and spacecraft are at that position.

The associated Legendre function is calculated according to a specific degree and order, *n* and *m*. The coefficients for zonal harmonics, or when $m = 0$, are solved using

$$a_{nm} = \frac{1}{MR_b^{\ 2}}\int\frac{\rho^2}{2}\left(3\sin\left(\phi'\right)-1\right)dM \quad (9)$$

Lastly, the coefficients for when $n = m$ or $n \neq m$, are calculated differently. These sectorial and tesseral harmonics use

$$a_{nm} = \frac{2(n-m)!}{MR_b^{\ n}(n+m)!}\int\rho^n P_n^m\left(\sin\left(\phi'\right)\right)\cos\left(m\theta'\right)dM \quad (10)$$

$$b_{nm} = \frac{2(n-m)!}{MR_b^{\ n}(n+m)!}\int\rho^n P_n^m\left(\sin\left(\phi'\right)\right)\sin\left(m\theta'\right)dM \quad (11)$$

where

$$\rho = \sqrt{\xi^2+\eta^2+\zeta^2} \quad (12)$$

In Eqs. 9 to 12, *n* is the degree, *m* is the order of expansion, *M* is the mass of the spheroid, ϕ' is the latitude of the mass element *dM* in the body-fixed coordinates, θ' is the longitude of the mass element *dM*, and $\xi, \eta,$ and ζ are fixed coordinates to the reference body. A subfunction of gravmod calculates the approximate coefficients prior to the calculation of the potential.

Spherical harmonics may be best approximation for bodies without polyhedral models, but due to the infinite expansions over and underflow becomes an issue. Another drawback is the amount of time each value needs to be calculated. To combat these drawbacks, a final program was developed that assumes a near perfect ellipsoid and uses moments of inertia to solve for the potentials.

E. Brillouin Sphere Approximation

Simplified gravitational models can be produced by using approximated shapes and densities. By approximating the shape and density of a body, the potentials can be calculated very quickly with low chances of over and underflow, or when such programs lose accuracy due to too small or too large of numbers. However, this method requires two separate solutions, internal potentials and external potentials. The program also references the Brillouin sphere for calculation. The Brillouin sphere is the smallest sphere that encompasses the entire body [14]. The potentials are defined into two different categories that require two separate solutions, internal potentials that are inside the Brillouin sphere, and external potentials that are outside the Brillouin sphere. This paper utilizes the simplified equations for the external potentials from the Herrera-Sucarrat and Palmer paper [15].

Due to the computational competence required for the previous methods implemented, a final program was developed that assumes a tri-axial ellipsoid with constant density to reduce complexity. The program produces an ellipsoid that is encompassed in the corresponding Brillouin sphere as shown in Fig. 3.

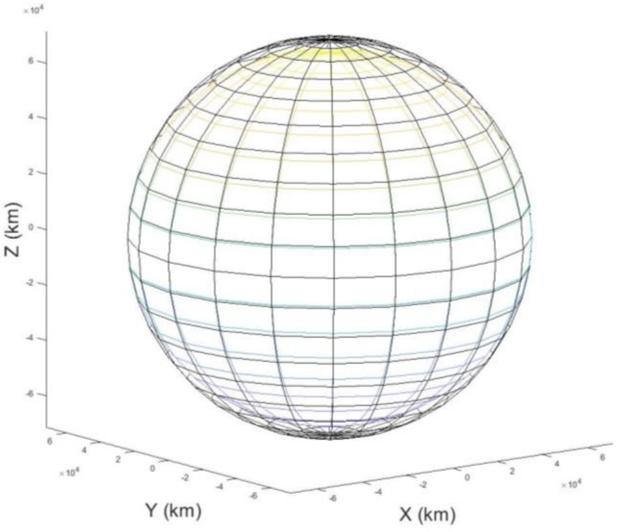

Fig. 3: Brillouin Sphere example using Jupiter and its rotational flattening

The colored spheroid in Fig. 3 is Jupiter and the black sphere is the Brillouin Sphere. Jupiter's largest radius is located on the z-axis at zero. Both spheres are centered directly on the origin, allowing easy calculation for the moments of the body. Currently only the potentials outside of the Brillouin Sphere can be calculated.

Provided the center of gravity is aligned with the origin of the coordinate system, the mass moments can be used to solve the potential up to the second order. If the coordinate system coincides with the origin, the mass moments of inertia are

$$I_{xx} = M_b\left(y^2 + z^2\right) \tag{13}$$

$$I_{yy} = M_b\left(x^2 + z^2\right) \tag{14}$$

$$I_{zz} = M_b\left(x^2 + y^2\right) \tag{15}$$

where *x*, *y*, and *z* are the radii lengths of the ellipsoid, and the products of inertia are zero. Using these inertias, the potential can be written as

$$U = \frac{GM}{\sqrt{x^2+y^2+z^2}}\left\{1 + \frac{1}{2\left(x^2+y^2+z^2\right)}\left[\frac{tr\left(I_3\right)}{M} - \frac{3}{x^2+y^2+z^2}\left(\frac{I_{xx}x^2 + I_{yy}y^2 + I_{zz}z^2}{M}\right)\right]\right\} \tag{16}$$

where G is the Gravitational constant, M is the mass of the body, I_3 is the diagonal matrix with elements I_{xx}, I_{yy}, I_{zz} and *x*, *y*, and *z* are the Cartesian coordinates of the spacecraft from the origin. This formula is known as MacCullagh's formula. The coordinates of the spacecraft must be outside of the Brillouin Sphere for the potentials to be calculated. Currently this function is being developed and tested in MATLAB but will be transitioned to Julia once completed.

III. Future Implementation

Along with the implementation of the polyhedron gravitational modeling, there are many opportunities for future improvement in regards to new features and additional compatibility. Modifications to be implemented in the near future include adding higher order spherical harmonics and expanding to ellipsoidal harmonics in the current Julia 1.1 version. Another expansion would be the ability to download and present asteroid shape files from the DAMIT database. Lastly, implementing a function for the geometric albedo for both major and small body objects. To improve current code functions, future work would also consist of optimizing gravmod, polymod, and astermod to reduce the run time and increase the computational efficiency. Another opportunity to increase boddat's capability is to implement features that enable compatibility with Spice (.bsp) files.

IV. Conclusion

Julia Language 1.1 Ephemeris Reader and Gravitational Modeling Program for Solar System Bodies was created to simplify and aid in future space missions as well as implement the Julia Language within NASA. The ephemeris reader works by accessing multiple NASA databases and directories to identify and retrieve the user requested values to assist in space mission planning. Unlike other ephemeris readers, boddat is able to retrieve information on both major and small bodies. This feature, in conjunction with the astermod function, provides the user with a wider spectrum for mission planning by granting access to asteroid properties and states. Additional functions were implemented to calculate the gravitational field of both spherical and non-spherical bodies. These functions allow the user to develop any asteroid related missions such as reconnaissance or rendezvous-based missions. The third-generation reader is also compatible with Julia 1.1. The Julia team has incorporated backwards compatibility after Julia version 1.0; therefore, the current reader is compatible with every version of Julia that is released after 1.0. The authors are in the process of making the current version of the package open-source and available to the public. The ephemeris reader will be an easy to use tool, for all users, that will assist in the astrodynamics aspect of preliminary space mission design.

Acknowledgments

This research was partially supported by the Embry-Riddle Aeronautical University (ERAU) Undergraduate Research Institution and the ERAU Faculty Research Development Program as well as Arizona Space Grant. A portion of this research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. Authors Tristan, Parker, and Brennan thank Kaela Martin and Damon Landau for their mentorship and guidance over the course of this project. The Authors Tristan, Parker, and Brennan also thank the Undergraduate Research Institution administrators Anne Boettcher and Virginia MacGowan for their support.

References

[1] Vasile, M., Ceriotti, M., Becerra V. M., and Nasuto, S. J., *Computational Intelligence in Aerospace Sciences*, American Institute of Aeronautics and Astronautics, Virginia, 2014, Ch. 20.

[2] Englander, J. A., and Conway, B. A., "Automated Solution of the Low-Thrust Interplanetary Trajectory Problem," *Journal of Guidance, Control, and Dynamics,* Vol. 40, No. 1, 2017, pp. 15-27.

[3] Stoneking, E. T., Neerav, S., and Dean, C. J., "Real-Time Visualization of Spacecraft Telemetry for the GLAST and LRO Missions," *SpaceOps Conference*, Huntsville, AL, 2010.

[4] Holdaway, R., "Satellite Ephemeris Prediction and Orbit Maneuverability by Low Thrust," *AIAA Electric Propulsion Conference*, New Orleans, LA 1975.

[5] Bezanson, J., Edelman, A., Karpinski, S., and Shah, V. B., "Julia: A Fresh Approach to Numerical Computing," *Society for Industrial and Applied Mathematics*, Vol. 59, No. 1, 2017, pp. 65-98.

[6] Martin, K., Mihaylov, J., Spear, R., and Landau, D., "Julia Language Ephemeris and Physical Constants Reader for Solar System Bodies," *AIAA SciTech 2018,* Kissimmee, FL, 2018.

[7] Martin, K., Mihaylov, J., Spear, R., and Landau, D., "Julia Language 1.0 Ephemeris and Physical Constants Reader for Solar System Bodies," *AAS 2019*, Ka'anapali, Maui, HI, 2019.

[8] Peabody, P. R., Scott, J. F., and Orozeo, E. G., “JPL Ephemeris Tapes E9510, E9511, and E9512,” Technical Memorandum 33-167. Jet Propulsion Laboratory, Pasadena, CA, 1964.

[9] Devine, C. J., “JPL Development Ephemeris Number 19,” Technical Report 33-1181. Jet Propulsion Laboratory, Pasadena, CA, 1967.

[10] Lieske, J, H., “Newtonian Planetary Ephemerides 1800-2000: Development Ephemeris Number 28,” *National Aeronautics and Space Administration*, Pasadena, 1967.

[11] Chesley, S. R., D., Farnocchia, D., Nolan, M. C., Vokrouhlicky, D., Chodas, P. W., Milani, A., Spoto, F., Rozitis, B., Benner, L. A. M., Bottke, W. F., Busch, M. W., Emery, J. P., Howell, E. S., Lauretta, D. S., Margot, J.-L., and Taylor, P. A., “Orbit and bulk density of the OSIRIS-REx target Asteroid (101955) Bennu,” *Icarus*, Vol. 235, 2014, pp. 5–22.

[12] Werner, R., and Scheeres, D., “Exterior gravitation of a polyhedron derived and compared with harmonic and mascon gravitation representations of asteroid 4769 Castalia,” *Celestial Mechanics and Dynamical Astronomy*, Vol. 65, No. 3, 1996, pp. 313-344.

[13] Takahashi, Y., and Scheeres, D. J., “Small Body Surface Gravity Fields via Spherical Harmonics Expansions,” *AIAA/AAS Astrodynamics Specialist Conference*, San Diego, CA, 2014.

[14] Relmond, S., and Baur, O., “Spheroidal and ellipsoidal harmonic expansions of the gravitational potential of small solar system bodies. Case study: Comet 67P/Churyumov-Gerasimenko,” *Journal of Geophysical Research Planets*, Vol. 121, No. 3, 2016, pp. 497-515.

[15] Herrera-Sucarrat, E., Palmer, P.L., and Roberts, R.M., “Modelling the gravitational potential of a non-spherical asteroid,” *Journal of Guidance, Control, and Dynamics*, Vol. 36, No. 3, 2013, pp. 790-798.